Selective Program Execution

Quick Start

Compile step

```shell
mkdir labs
javac Flooring2.java
```

Execute step

```shell
cd labs
mkdir 2
```

```shell
cd 2
java Flooring2
```

```shell
cp /samples/csc/156/labs/2/* .
```

Submit step

```shell
emacs Flooring2.java & submit csc156abc 2
```

Problem Statement and Specifications

This assignment is unique in that we will be searching for a variety of boolean conditions as our program executes and will issue a different output statement for each one. This will provide evidence to the user as to the outcome of the program’s execution.

Assignment 2 Statement:
Write a program in a file named Flooring2.java that provides a user with a menu of options that describe 3 types of flooring. When the user inputs an appropriate choice, display a message that indicates the flooring type that they have chosen. Be sure to accommodate the possibility that the user mistakenly enters a non-menu value.

During this algorithm, if there are \( n \) options in the menu, there are then \( n + 1 \) possible outcomes that can occur. Consequently, amongst multiple executions we will have to check each boolean condition to completely test out our program. Therefore, our output must consist of the informational message that indicates which of the \( n + 1 \) outcomes has occurred. Thus, we have the following specifications.

Assignment 2 Specifications:
Input  
Flooring selection choice

Output  
A message indicating the unique output associated with that selection choice

Object Analysis and Algorithmic Development

We will be interested in displaying String values as informational objects in our output statements. Our input menu choice can be either int or char. There will also be a variety of boolean expressions deciding the outcomes of our selection statements, however there is no immediate need for variables of this type. We’ll again start the process by analyzing the simplest useful algorithm that we’ve seen before.
1. Input

2. Process

3. Output

The only input value for this program is the choice indicating the menu value that we’re interested in. Consequently, we’ll need to prompt for this value with the menu itself. Then, we’ll need to determine the output message associated with the choice that was input.

1. output a menu of options

2. input a choice selected from the menu

3. process to determine the appropriate message based upon the menu choice

4. output the appropriate message

Since we have an option of using either int or char for the choice and we’ll be using JOptionPane.showInputDialog() methods for input, there will need to be some type of conversion or access from String for the underlying int or char value that is identified as a menu selection. We’ll provide details as we follow the pseudo code below. Meanwhile, we’ll simply refer to that step as a conversion for testing on the choice. Lines that are displayed in blue represent areas where you will need to write or modify code to complete the assignment.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) display a descriptive message of the program</td>
<td>String constants</td>
</tr>
<tr>
<td>2) display a menu of options for flooring</td>
<td>String constants &amp; variable</td>
</tr>
<tr>
<td>3) input a choice based upon the menu</td>
<td>int or char variable &amp; String variable</td>
</tr>
<tr>
<td>4) determine which of 4 output statements should be displayed</td>
<td>int or char variable, boolean expressions &amp; String variable &amp; constants</td>
</tr>
<tr>
<td>5) display appropriate message</td>
<td>String constants &amp; variables</td>
</tr>
</tbody>
</table>

Coding for Compilation

Login to your csc.oakton.edu account and create a new subdirectory of the labs directory called 2 to work on your assignment.

cd labs
mkdir 2
cd 2

The documentation for this algorithm has already been included in the file /samples/csc/156/labs/2/Flooring2.java that you can copy over to your local directory by issuing the following cp command.

    cp /samples/csc/156/labs/2/Flooring2.java .
    emacs Flooring2.java &

Declaration of Variables

As was mentioned previously, the variable selector will need to be input from a JOptionPane.showInputDialog() method and so it must be String. All output values will need to be String, but in this problem there is no need for formatting of output.
Follow the Pseudo Code

Step 1

    // Display a descriptive message of the program

This is simply an output statement that displays the message shown in Figure 1. Code this statement in the manner that you displayed output in the previous labs.

Step 2

    // Display a menu of options for flooring

This is nothing more than an output statement that prompts the user for the variable selector, such as that shown in Figure 2.

```
    String selector = JOptionPane.showInputDialog("Enter the number that ...");
```

The output of this statement is displayed in Figure 2. Code these statements into your program before continuing on.
// Input a choice based upon the menu

This input that occurred in to a String variable must be converted into either a char or int expression. In the case of char data, we would only want the first char value entered at the keyboard. This would be available to us by the following expression.

```
char choice = selector.charAt(0);
```

In the case of int data, we would need to access the parseInt() method of the Integer wrapper class. We’ll assume that this is the manner in which we’d like to proceed and provide options on the char strategy later. Consequently, the statement that we’ll need is the following.

```
int choice = Integer.parseInt(selector);
```

Steps 4-5

// determine which of 4 output statements should be displayed
// display appropriate message

Because our algorithm has 4 possible outcomes, we’ll need a means to select the appropriate output based upon the value of the variable choice. There are two options available to us that we will survey over this and the next assignment. For this exercise, we’ll use the binary selection statement. The basic syntax for a binary selection (if) statement in this language is the following.

```
if (boolean expression)
{
    statementT1;
    statementT2;
    ...
    statementTn;
}
else
{
    statementF1;
    statementF2;
    ...
    statementFm;
}
```

Essentially, statementT1, ..., statementTn are executed when the boolean_expression is true and statementF1, ..., statementFm are executed when the boolean_expression is false. The braces {} are not necessary when either m = 1 or n = 1. In addition, the else clause is not necessary. That is, we can also code what is sometimes referred to as the elseless if.

```
if (boolean_expression)
{
    statementT1;
    statementT2;
```

```
This first form is the one that we’ll be using in our program. Let’s restrict our example for now to assume that there are two possible outcomes: either the user enters one of the valid menu choices or not. To extend this analysis, let’s assume that the message in Figure 2 simply offers the choice of message 1 as the valid choice and we’re interested in displaying some type of error message should the user enter something other than 1. Let’s focus on getting this much working and then we’ll simply extend the code to ensure that 2 and 3 are valid for variable choice as well. One option that we’ll discuss in pseudo code for now would be to print a different output statement in each case. Something like the following would suffice.

```
if choice equals 1
    display "You chose selection 1 - Scored concrete, costs $3000"
else
    display "Your selection is not available"
```

Another option, and one that is easier to extend to multiple cases, is to create a String variable that is assigned different values within the different logical cases in the selection process. Then, we simply code one output statement after the assignments have occurred. That pseudo-code would look as follows.

```
if choice equals 1
    assign to output_string "You chose selection 1 - Scored concrete, costs $3000"
else
    assign to output_string "Your selection is not available"
    display output_string
```

To see this second strategy in Java code, we would use the following.

```
String output_string = null;
if (choice == 1)
    output_string = "You chose selection 1 - Scored concrete, costs $3000";
else
    output_string = "Your selection is not available";
JOptionPane.showMessageDialog(null, output_string, "Flooring Selection",
                JOptionPane.INFORMATION_MESSAGE);
```

The results of this code would be to display the output that is shown in Figure 3 when choice == 1 and the output that is shown in Figure 4 otherwise.

These steps might appear radically different should we be comparing data of type double, and not type int. The reason that this is important has to do with round off or truncation errors in real expressions. It is certainly the case that mathematically, $3.0/\sqrt{3} = e^{(\ln 3)/2}$. However, due to algorithmic issues and truncation error, it may not necessarily be true that $3.0/Math.sqrt(3.0) == Math.exp(0.5*Math.log(3.0))$. Consequently, whenever we make comparisons of real expressions, we should do so by ensuring that the absolute value of their difference is small. For our purposes, small could be considered to be $10^{-5}$. Consequently, the test for equality of double variables a and b should not be written as the following.
if (a == b) {
    ...
}

Instead, the selection statement should be coded in the following manner.

if (Math.abs(a - b) < 1e-5) {
    ...
}

Checkpoint

Although we’re not done, at this point, it might be a good idea to compile this much before we build more. As with our previous labs, the idea of checkpoint activities is to try and discover our errors before we generate too much code.

Save your program to disk by choosing the Save command from the Files menu of your emacs session. Then, compile your program by choosing the Compile... command from the Tools menu and change the make -k that is displayed to javac Flooring2.java. Compiler errors can be parsed with the keystroke C-x ‘ and need to be repaired before your program can execute. When your program has compiled, click on your xterm window to access your command line prompt, and issue the command java Flooring2.

Steps 4-5 (again)

// determine which of 4 output statements should be displayed
// display appropriate message

All that needs to be done now is to extend the logic to include the other cases. Consequently, we’d like to code the following statements into our program.

```java
String output_string = null;
```
```java
if (choice == 1)
    output_string = "You chose selection 1 - Scored concrete, costs $3000";
else if (choice == 2)
    output_string = "You chose selection 2 - Carpeting comes with a $5000 allowance";
else if (choice == 3)
    output_string = "You chose selection 3 - Wood floors in the living area
        + carpeting in the bedrooms, tile in the bath areas
        + and a $5000 carpet allowance, totaling $10,000"
else
    output_string = "Your selection is not available";

JOptionPane.showMessageDialog(null, output_string, "Flooring Selection",
                        JOptionPane.INFORMATION_MESSAGE);
```

So now, the results of would include the output that is shown in Figure 5 when `choice == 2` and the output that is shown in Figure 6 when `choice == 3`. We’ll note that if the variable `choice` were of type `char` instead of `int`, these comparisons would have looked like `choice == '1'`, `choice == '2'` and `choice == '3'`.

**Testing for errors**

You need to save your program, compile and execute it as you did up above at the Checkpoint. If you have any logic errors, they probably are limited to the code that you wrote to accomplish Step 4. There is a correctly functioning version of `Flooring2.java` at this link.

**Printing and submitting**

Once you are satisfied with the correctness of your program, print it as you did with previous assignments by using the following command that assumes that you are working in the room 1234 at Oakton. Retrieve your copy from the printer.

```
printer 1234 Flooring2.java
```
Finally, submit your program with the following command that assumes that you are registered in section abc of CSC 156.

```
submit csc156 abc 2
```